Sensitivity Direction Learning with Neural Networks Using Domain Knowledge as Soft Shape Constraints

Kazuyuki Wakasugi
Mitsubishi Heavy Industries
kazuyuki_wakasugi@mhi.co.jp

Abstract

If domain knowledge can be integrated as an appropriate constraint, it is highly possible that the generalization performance of a neural network model can be improved. We propose Sensitivity Direction Learning (SDL) for learning about the neural network model with user-specified relationships (e.g., monotonicity, convexity) between each input feature and the output of the model by imposing soft shape constraints which represent domain knowledge. To impose soft shape constraints, SDL uses a novel penalty function, Sensitivity Direction Error (SDE) function, which returns the squared error between coefficients of the approximation curve for each Individual Conditional Expectation plot and coefficient constraints which represent domain knowledge. The effectiveness of our concept was verified by simple experiments. Similar to those such as L2 regularization and dropout, SDL and SDE can be used without changing the neural network architecture. We believe our algorithm can be a strong candidate for neural network users who want to incorporate domain knowledge.

1 Introduction

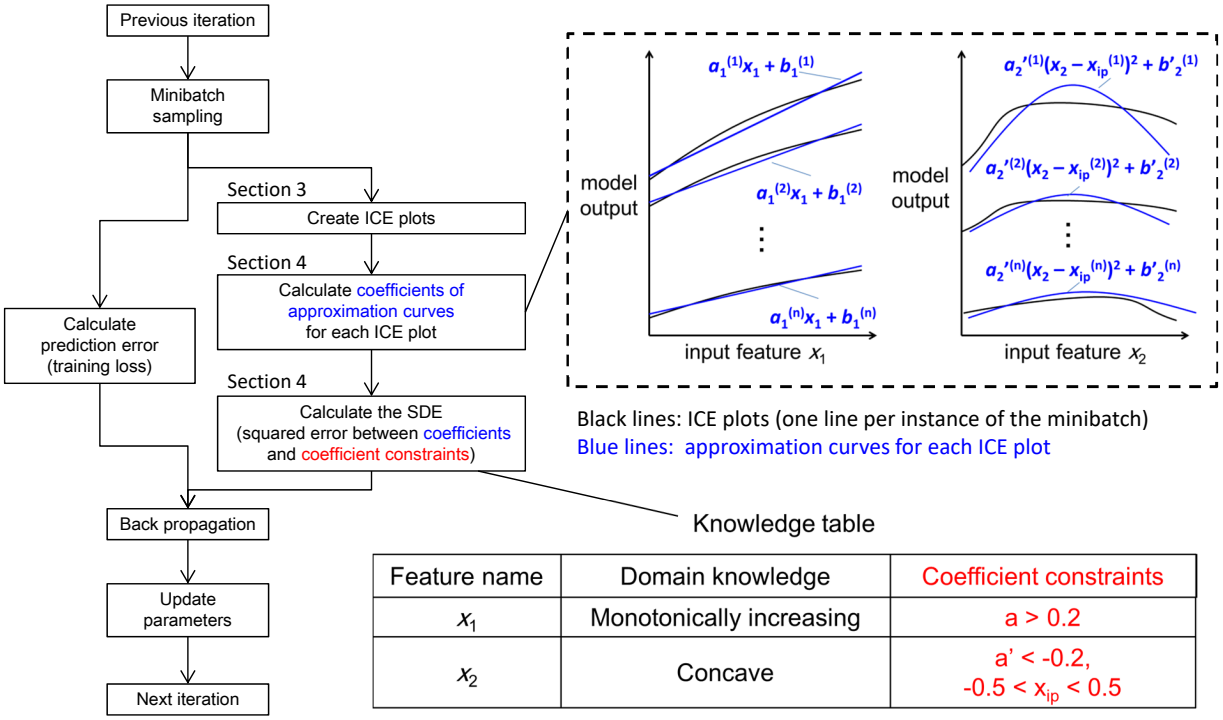
As neural networks have been used in many applications, it has become a non-negligible issue that, if the model's generalization performance is low, the predictions of the model can cause inappropriate and dangerous decision making [Gill and Hall, 2019]. In order to prevent the above issue, there is a well-known approach [Molnar, 2019] which visualizes the relationships between each input feature and the output of the model using XAI algorithms such as Partial Dependence (PD) [Friedman, 2001] and Individual Conditional Expectation (ICE) [Goldstein *et al.*, 2015] plots and judges whether approximate shapes of these plots match domain knowledge (e.g., monotonicity, convexity). However this approach can only be used for judgment, and it is not an easy task to learn about the model with relationships matched to domain knowledge especially if there is bias in the data collection.

There are several existing solutions to learn about a neural network model with user-specified relationships even under the condition mentioned in the previous paragraph. These algorithms can be divided into three categories: (1) imposing hard monotonicity constraints by changing the model architecture [Archer and Wang, 1993; Sill, 1998; Daniels and Velikova, 2010; You *et al.*, 2017; Wehenkel and Louppe, 2019], (2) imposing hard convexity constraints by changing the model architecture [Amos *et al.*, 2017; Gupta *et al.*, 2018], and (3) imposing soft monotonicity constraints by changing the learning process [Gupta *et al.*, 2019].

Based on previous research, we develop an algorithm, Sensitivity Direction Learning (SDL), which can impose not only monotonic constraints but also various kinds of constraints such as convexity by changing the learning process. The motivations for our research are: (1) convexity constraints should also be able to be imposed, because there are many applications where the output is a convex function of the input such as air temperature and electric power demand [McMenamin, 2008], frequency and impedance [Nguyen and Howe, 1992], various characteristic curves [Horváth *et al.*, 2019; Mulders *et al.*, 2020], etc., and (2) changing the learning process has the advantage that it can be used universally for the existing neural network architecture.

An overview of SDL is shown in Figure 1. Characteristics of the learning process of SDL are (1) to calculate not only the conventional prediction error (such as mean squared error) but also calculate Sensitivity Direction Error (SDE), and (2) to train the neural network model to minimize both the prediction error and SDE. SDE is calculated by the SDE function, which returns the squared error between coefficients of the approximation curve for each ICE plot and coefficient constraints which represent domain knowledge. Since low SDE means that approximate shapes of each ICE plot matches domain knowledge, the generalization performance of the low SDE model expects to be superior to that of high SDE models if prediction errors of the models are the same level. In this paper, five types of relationships ($Increase$, $Decrease$, $Convex$, $Concave$, and $Negligible$), which represent domain knowledge, are described. Furthermore, it can be applicable to other relationships where domain knowledge can be expressed as coefficient constraints. We call these approximate relationships **sensitivity directions** and these domain knowledge **knowledge table**, which consists of sensitivity directions and values of coefficient constraints for each feature.

We ran two experiments. In experiment 1, it was confirmed that SDL actually learned neural network models with



Feature name	Domain knowledge	Coefficient constraints
x_1	Monotonically increasing	a > 0.2
x_2	Concave	a' < -0.2, -0.5 < x_{ip} < 0.5

Figure 1: An overview of Sensitivity Direction Learning (SDL). In this figure, the flowchart describes how SDL works in one iteration during the neural network training. Upper right figures show an example of ICE plots and approximation curves for each ICE plot. Lower right table shows an example of a **knowledge table** consist of user-specified relationships (we call these approximate relationships **sensitivity directions**) and coefficient constraints which represent domain knowledge.

Approach	Target relationship: Only monotonicity	Target relationship: Including convexity
Model architecture	MN [1998], MNN [2010], UMNN [2019], etc.	ELN [2016], ICNN [2017], DLN [2018], etc.
Learning process	PWL [2019]	**This paper**

Table 1: Positioning of our research.

five types of user-specified relationships using the generated dataset. In experiment 2, using three actual datasets, it was confirmed that SDL has the potential to achieve the same or better prediction performances than five kinds of baseline machine learning algorithms (three non-shape-constrained and two shape-constrained algorithms).

2 Related Work

The first algorithm which can impose hard monotonicity constraints is to positively or negatively limit the neural network weights using single-layer neural networks [Archer and Wang, 1993]. Monotonic Networks (MN) [Sill, 1998] and Monotone Neural Networks (MNN) [Daniels and Velikova, 2010] extended to using multiple layers. Unconstrained Monotonic Neural Networks (UMNN) [Wehenkel and Louppe, 2019] used monotonic activation functions. In addition, Input Convex Neural Networks (ICNN) [Amos *et al.*, 2017] can impose hard convexity constraints by using unique architectures such as convex activation functions and passthrough layers, and it was extended to the Shape-Constrained Neural Network (SCNN) [Gupta *et al.*, 2018] which uses ReLU activation functions instead of convex activation functions. Ensemble lattice networks (ELN) [Canini *et al.*, 2016] and Deep lattice networks (DLN) [You *et al.*, 2017] can impose hard monotonicity and also convexity [Gupta *et al.*, 2018] constraints by using unique architectures such as a linear calibrator and an ensemble of lattices. These algorithms have characteristics that can impose hard monotonicity and/or convexity constraints by changing the model architecture. On the other hand, soft monotonicity constraint algorithms that change the learning process were also proposed. One of these algorithms uses a point-wise loss (PWL) function [Gupta *et al.*, 2019] which returns the error between the partial derivatives of the output of the model with respect to each input feature and constraints for the sign of each partial derivative.

As mentioned in Section 1, we consider that (1) convexity constraints should also be able to be imposed, and (2) changing the learning process has the advantage that it can be used universally for the existing neural network architecture. However as far as we know, there are no studies that satisfy both factors. Therefore, we believe that our research contributes to the field as shown in Table 1.

3 Preliminaries

Our algorithm (SDL) uses Individual Conditional Expectation (ICE) plots. This section will briefly discuss ICE plots.

For ease of understanding, we will start with the Partial Dependence (PD) plot which is the previous study of the ICE plot. The PD plot is one of the model-agnostic methods that visualizes the relationship between the output of the model and one input feature based on Equation 1.

$$\bar{f}(x_k) = E_{X_c}\left[\hat{f}(x_k, X_c)\right] = \frac{1}{n}\sum_{i=1}^{n}\hat{f}(x_k, X_c^{(i)}) \quad (1)$$

Where x_k is the value of one input feature to visualize the relationship, and $\hat{f}$ is the trained model. $X_c^{(i)}$ is the 1-d vector of all input features of the model without x_k, and n is the sample number of instances in the given dataset and $\bar{f}(x_k)$ is the partial dependence, which means averaging outputs of the model for each instance in the given dataset while fixing the value of x_k. Therefore, by checking PD plots, the approximate shape of the relationship between the output of the model and target input feature can be evaluated.

The ICE plot can visualize the relationships between x_k and each instance of $\hat{f}(x_k, X_c^{(i)})$. Compared to checking PD plots, checking ICE plots is especially important when approximate shapes of each ICE plot are very different.

4 Sensitivity Direction Learning

Our proposal can be divided into four parts:

1. Variable definition of coefficient constraints that are required to evaluate whether approximate shapes of each ICE plot match user-specified relationships.
2. The estimation algorithm about coefficients of the approximation curve for each ICE plot of the model.
3. The SDE function which returns the squared error between coefficients of the approximation curve and coefficient constraints which represent domain knowledge.
4. The learning process of SDL during one iteration.

4.1 Variable Definition of Coefficient Constraints

Coefficient constraints mean such as upper and lower limits for coefficients of the approximation curve for each ICE plot. We define variables of coefficient constraints for judging the following five types of sensitivity directions. In addition, the following definition process can be applicable to other kinds of sensitivity directions.

Monotonically Increasing

$Increase$ indicates that as the input feature value x increases, the output value y also increases. To judge for $Increase$, the required constraint variable is the lower limit $\boldsymbol{a_{\min}}$ of the inclination $\boldsymbol{a}$ of the approximate straight line $y \approx ax$.

Monotonically Decreasing

$Decrease$ is the opposite of $Increase$, so that the required constraint variable is the upper limit $\boldsymbol{a_{\max}}$.

Convex

$Convex$ indicates that the initial behavior is $Decrease$, and it changes to $Increase$ in the middle. To judge for $Convex$, it is necessary to know the inclination a' of the approximate straight line $dy/dx \approx a'x$, and the x coordinate of the inflection point x_{ip}. Therefore, required constraint variables are lower limit $\boldsymbol{a'_{\min}}$ of inclination $\boldsymbol{a'}$ and upper and lower the limits ($\boldsymbol{x_{\mathrm{ip}_{\max}}}$, $\boldsymbol{x_{\mathrm{ip}_{\min}}}$) of $\boldsymbol{x_{\mathrm{ip}}}$.

Concave

$Concave$ is the opposite of $Convex$, so required constraint variables are the upper limit $\boldsymbol{a'_{\max}}$ of inclination $\boldsymbol{a'}$ and upper and lower limits ($\boldsymbol{x_{\mathrm{ip}_{\max}}}$, $\boldsymbol{x_{\mathrm{ip}_{\min}}}$) of $\boldsymbol{x_{\mathrm{ip}}}$.

Negligible

$Negligible$ indicates that the absolute value of the inclination is very small, so that required constraint variables are upper and lower limits ($\boldsymbol{a_{\max}}$, $\boldsymbol{a_{\min}}$).

Summarizing the above arguments, required constraint variables are following: (1) $a_{\min}$, (2) $a_{\max}$, (3) $a'_{\min}$, (4) $a'_{\max}$, (5) $x_{\mathrm{ip}_{\min}}$, and (6) $x_{\mathrm{ip}_{\max}}$. We also define the sensitivity direction type t_{sd} such as $Increase$, $Decrease$, $Convex$, $Concave$, and $Negligible$.

4.2 Estimation Algorithm for Constraint Variables

This subsection presents an algorithm for estimating the three coefficients (a, a', x_{ip}) from each ICE plot.

Putting it simply, this algorithm applies single regression analysis on ICE plots as follows:

$$a_k^{(i)} = \frac{\left\{\hat{F}\left(X_k, \boldsymbol{X}_c^{(i)}\right) - \frac{1}{m}\sum_{j=1}^{m}\hat{f}\left(x_{k,j}, \boldsymbol{X}_c^{(i)}\right)\right\}\cdot X_k}{X_k \cdot X_k} \quad (2)$$

$$a'^{(i)}_k = \frac{\left\{\hat{F}'\left(X_k, \boldsymbol{X}_c^{(i)}\right) - \frac{1}{m}\sum_{j=1}^{m}\hat{f}'\left(x_{k,j}, \boldsymbol{X}_c^{(i)}\right)\right\}\cdot X_k}{X_k \cdot X_k} \quad (3)$$

$$x_{\mathrm{ip_k}}^{(i)} = -\frac{\sum_{j=1}^{m}\hat{f}'\left(x_{k,j}, \boldsymbol{X}_c^{(i)}\right) - a'^{(i)}_k\sum_{j=1}^{m}x_{k,j}}{m a'^{(i)}_k} \quad (4)$$

Where, $x_{k,j}$ is one of arbitrary m values of x_k for drawing ICE plots, and X_k is the 1-d vector that is composed of each $x_{k,j}$ ($X_k = \{x_{k,1}, x_{k,2}, \cdots, x_{k,m}\}$). Likewise, $\hat{F}\left(X_k, X_c^{(i)}\right)$ is the 1-d vector composed of each $\hat{f}\left(x_{k,j}, X_c^{(i)}\right)$ and $\hat{F}'\left(X_k, X_c^{(i)}\right)$ is the 1-d vector composed of each of $\hat{f}'\left(x_{k,j}, X_c^{(i)}\right)$. $\hat{f}'$ is the differential function of the $\hat{f}$ by x_k. Outputs of the estimation algorithm are three matrices (A_k, A'_k, and $X_{\mathrm{ip_k}}$) shown in $A_k^T = \left\{a_k^{(1)}, a_k^{(2)}, \cdots, a_k^{(n)}\right\}$, $A'^T_k = \left\{a'^{(1)}_k, a'^{(2)}_k, \cdots, a'^{(n)}_k\right\}$, and $X^T_{\mathrm{ip_k}} = \left\{x^{(1)}_{\mathrm{ip_k}}, x^{(2)}_{\mathrm{ip_k}}, \cdots, x^{(n)}_{\mathrm{ip_k}}\right\}$.

Algorithm 1 calcSDE

Input: A_k, A'_k, $X_{\mathrm{ip_k}}$
Input: $N_k : [t_{\mathrm{sd}}, a_{\max}, a_{\min}, a'_{\max}, a'_{\min}, x_{\mathrm{ip}_{\max}}, x_{\mathrm{ip}_{\min}}]$
Output: $e_{\mathrm{sd_k}}$ # SDE of feature k

if t_{sd} is $Increase$ or $Decrease$ or $Negligible$ **then**
  $e_{\mathrm{sd_k}} \leftarrow \mathrm{calcDev}(A_k, a_{\max}, a_{\min})$
else if t_{sd} is $Convex$ or $Concave$ **then**
  $e_{\mathrm{sd_k}} \leftarrow \mathrm{calcDev}(A'_k, a'_{\max}, a'_{\min})$
  $e_{\mathrm{sd_k}} \leftarrow e_{\mathrm{sd_k}} + \mathrm{calcDev}(X_{\mathrm{ip_k}}, x_{\mathrm{ip}_{\max}}, x_{\mathrm{ip}_{\min}})$
else
  $e_{\mathrm{sd_k}} \leftarrow 0.0$
end if

Algorithm 2 calcDev

Input: X, $x_{\max}$, $x_{\min}$
Output: d # Constraint violation

$d \leftarrow 0.0$
for $i = 1 : n$ **do**
  $x \leftarrow X[i]$
  $d \leftarrow d + \{\mathrm{MAX}(x, x_{\max}) - x_{\max}\}^2$ # Violating a upper constraint
  $d \leftarrow d + \{x_{\min} - \mathrm{MIN}(x, x_{\min})\}^2$ # Violating a lower constraint
end for
$d \leftarrow d/n$

4.3 SDE Function

The SDE function returns the squared error between coefficients (A_k, A'_k, and $X_{\mathrm{ip_k}}$) and coefficient constraints N_k ($N_k = \{t_{\mathrm{sd}}, a_{\max}, a_{\min}, a'_{\max}, a'_{\min}, x_{\mathrm{ip}_{\max}}, x_{\mathrm{ip}_{\min}}\}$). The algorithm of SDE function is shown in Algorithm 1. Where, $e_{\mathrm{sd_k}}$ is the value of SDE of the feature k. $e_{\mathrm{sd_k}}$ is calculated by the calcDev function according to the sensitivity direction type t_{sd}. The calcDev function, as shown in Algorithm 2, calculates the constraint violation d by using the input 1-d vector X and upper and lower limits ($x_{\max}$, $x_{\min}$). for each element x in X. Therefore, a high $e_{\mathrm{sd_k}}$ indicates that the relationship between the input feature k and the output of the model is differ from the user-specified relationship.

4.4 Sensitivity Direction Learning Algorithm

We will explain how SDL works in one iteration during the neural network training as shown in Algorithm 3. Where, the training dataset $\boldsymbol{D}_T$ consists of l input features and one target, and $\boldsymbol{N}$ is the matrix composed of each N_k ($\boldsymbol{N}^T = \{N_1, N_2, \cdots, N_l\}$). Mixing ratio r is the hyperparameter used to adjust the priority of SDE (e_{sd}) over the prediction error (e_{p}), and $\boldsymbol{D}_x$ consists of all input features extracted from the minibatch, and the rest is the output vector D_y. L is the conventional loss function such as the mean squared error function. $\hat{f}_{\mathrm{NN}}$ is the neural network model, and $\boldsymbol{X}_c$ is the matrix composed of each $X_c^{(i)}$ ($\boldsymbol{X}_c^T = \left\{X_c^{(1)}, X_c^{(2)}, \cdots, X_c^{(n)}\right\}$).

In each iteration, SDL calculates the prediction error (e_{p}) between $\hat{f}_{\mathrm{NN}}(\boldsymbol{D}_x)$ and D_y, and also calculates SDE (e_{sd}).

Algorithm 3 Sensitivity Direction Learning (SDL)

Input: Training dataset $\boldsymbol{D}_T$, Knowledge table $\boldsymbol{N}$
Parameter: Mixing ratio r

for Each minibatch ($\boldsymbol{D}_x$, D_y) sampled from $\boldsymbol{D}_T$ **do**
  $e_{\mathrm{p}} \leftarrow L\left(\hat{f}_{\mathrm{NN}}(\boldsymbol{D}_x), D_y\right)$ # Prediction error
  $e_{\mathrm{sd}} \leftarrow 0.0$ # Intialize SDE
  for $k = 1 : l$ **do**
    $N_k \leftarrow \boldsymbol{N}[k]$
    $\boldsymbol{X}_c \leftarrow \boldsymbol{D}_x$ without k
    $X_k \leftarrow$ Aquire m samples from the domain of x_k
    A_k, A'_k, $X_{\mathrm{ip_k}} \leftarrow \boldsymbol{X}_c$, X_k and Equation $1 \sim 4$
    $e_{\mathrm{sd}} \leftarrow e_{\mathrm{sd}} + \mathrm{calcSDE}(A_k, A'_k, X_{\mathrm{ip_k}}, N_k)$
  end for
  $e \leftarrow e_{\mathrm{p}} + r \cdot e_{\mathrm{sd}}$
  Update parameters with e
end for

And next, updates parameters of the neural network to reduce e which is the sum of e_{p} and $r \cdot e_{\mathrm{sd}}$. In the loop of each feature k, firstly SDL extracts N_k from $\boldsymbol{N}$ and $\boldsymbol{X}_c$ from $\boldsymbol{D}_x$, secondly extracts X_k which consists of m values from the domain of x_k, and finally, SDL calculates SDE of feature k based on Algorithm 1. SDL minimizes both the prediction error and SDE, so that the neural network model with user-specified relationships can be obtained.

5 Experiments

We ran two experiments to answer the following basic research questions:

- Whether does SDL actually work to learn about the model with user-specified directions?
- Under what conditions is SDL effective in improving the generalization performance?

5.1 Experiment 1

In this experiment, we confirmed that SDL actually works to learn about the model with user-specified sensitivity directions using the generated dataset.

Dataset

In order to clearly confirm that SDL actually works to learn about the model with user-specified relationships, the dataset should be have relationships differed from user-specified relationships. Considering this requirement, we generated the dataset which consists of 200 instances and six columns as following steps: (i) the 1-d vector U is defined to be between 0.0 and 1.0 in 0.005 steps, and (ii) y is set to U, and each feature ($x_1 \sim x_5$) is set to the sum of U and Gaussian noise ($\sigma = 0.1$).

As is clear from above steps, there are only strong correlations between each of five features ($x_1 \sim x_5$) and the output y. Therefore it can be said that if a model is learned without shape constraints, there is very little possibility that the trained model has convexity relationships.

Feature name	Sensitivity direction type	Coefficient constraints
x_1	$Convex$	$0.18 < a'$, $-0.58 < x_{\mathrm{ip}} < 0.58$
x_2	$Concave$	$a' < -0.75$, $-0.31 < x_{\mathrm{ip}} < 0.31$
x_3	$Increase$	$0.53 < a$
x_4	$Decrease$	$a < -0.64$
x_5	$Negligible$	$-0.08 < a < 0.08$

Table 2: One knowledge table out of 10 in experiment 1.

Mixing ratio	SDE (Med.)	SDE (SD)
0.0 (Vanila NN)	$8.54 * 10^3$	$4.02 * 10^6$
0.01	2.03	1.10
0.10	0.64	0.47
1.00	**0.24**	**0.35**
10.00	0.27	0.37

Table 3: Evaluation results of each mixing ratio.

Knowledge Table

In this experiment, 10 kinds of knowledge tables are tested. In each table, five sensitivity directions are set for each and every input feature and coefficient constraints are populated with random samples from a uniform distribution over $[0, 1)$ (only if the sensitivity direction is $Negligible$, from a uniform distribution over $[0, 0.1)$). As an example, one knowledge table is shown in Table 2.

Model Parameters

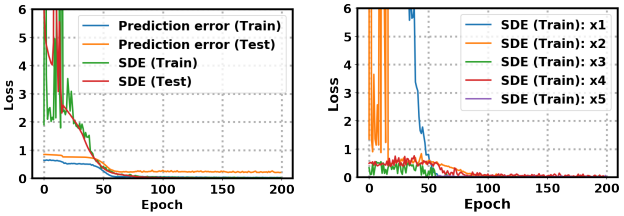
The neural network architecture and learning conditions are the same as in a Keras tutorial[1], as an example to show the general settings. On the other hand, the number of epochs is set to 200 after checking for learning curves.

In the SDL algorithm, the mixing ratio r, which used to adjust the priority of SDE over the prediction error, is the most important hyperparameter, so that five kinds of candidates ($\{0.0, 0.01, 0.10, 1.0, 10.0\}$) are tested.

Experimental Results

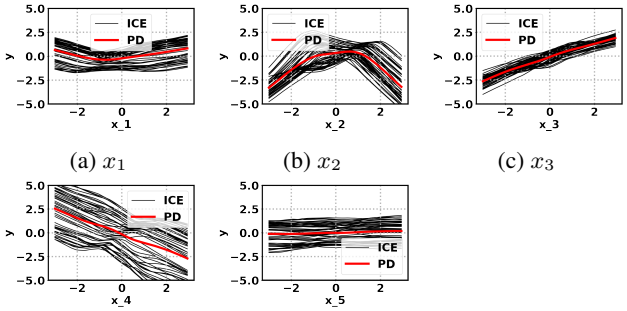
Table 3 shows medians and standard deviations of SDE for the test dataset for each mixing ratio. In this experiment, 50 instances were randomly extracted from the dataset as the test dataset, and the rest instances were used as the training dataset. Each median and standard deviation were calculated from each result for 10 different knowledge tables. As can be seen from this table, when the mixing ratio is 0.0, which means learning without SDL, the median of SDE was extremely worse. Therefore, it was confirmed that the low SDE was due to the SDL. In addition, the minimum median is 0.24 when the mixing ratio is 1.0, so that in this experiment, it is considered that the optimum mixing ratio value is around 1.0. This approach is considered to be generally applicable to the search for an appropriate mixing ratio.

[1]https://www.tensorflow.org/tutorials/keras/regression/



(a) Prediction error and SDE (b) SDE of each feature

Figure 2: Learning curves for SDL in experiment 1.



(a) x_1 (b) x_2 (c) x_3

(d) x_4 (e) x_5

Figure 3: PD and ICE plots in experiment 1.

Figure 2 and 3 show the results under the conditions that the mixing ratio is set to 1.0 and the knowledge table is used shown in Table 2. Figure 2a shows learning curves of the prediction error and SDE for the training and the test dataset. From Figure 2a, it can be seen that both the prediction error and SDE approximately converged at about 50 epochs. That is, it was confirmed that the two kinds of errors did not conflict. Figure 2b shows learning curves of SDE of each input feature. From this figure, SDE of each input feature ($x_1 \sim x_5$) all converged around $50 \sim 100$ epochs so that it can be said that SDL properly trained model with all of user-specified sensitivity directions shown in the knowledge table. Figure 3 shows PD and ICE plots of the trained model. Each ICE plot (50 black lines) corresponds to each instance of the test dataset and each PD plot (red line) is the average of black lines. From each figure, it can be said that sensitivity directions judged visually are as specified to Table 2.

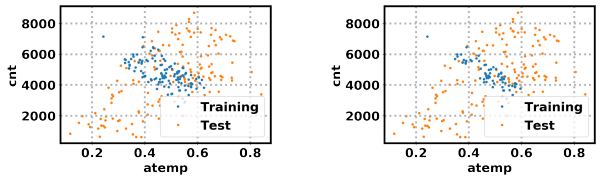
Therefore, it was confirmed that SDL actually worked to learn about models with user-specified sensitivity directions.

5.2 Experiment 2

In this experiment, using three actual datasets and 5-fold cross validation, we confirmed that SDL has the potential to provide the same or better prediction performances than five kinds of baseline algorithms, such as ELN, which is the one of the state-of-the-art shape-constrained algorithm. Furthermore, we will discuss when SDL can be effective.

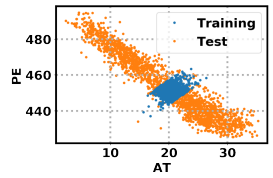
Dataset

Three actual datasets were chosen from the UCI Machine Learning Repository [Dua and Graff, 2017]. The first one is the Bike Sharing dataset [Fanaee-T and Gama, 2014] which

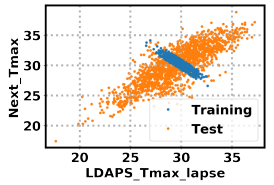


(a) Bike Sharing (qcut=5)

(b) Bike Sharing (qcut=9)



(c) CCPP (qcut=9)



(d) Bias correction (qcut=9)

Figure 4: Four examples of biased separation results in experiment 2. In each cross validation, the original training dataset is quantile-based discretized in the direction of the first principal component, and only the data at the center of the discretization is extracted as the biased training dataset. Five or nine for the number of divisions (**qcut**) are used. Principal component analysis is performed on two variables: the standardized output and one standardized input feature that has the highest absolute value of the correlation coefficient. The test dataset is used as-is without any changes.

consists of 731 instances and 11 features and has the characteristics that the output (count of total rental bikes (cnt)) may increase as the feeling temperature (atemp) increases. The second one is the Combined Cycle Power Plant (CCPP) dataset[Tüfekci, 2014] which consists of 9568 instances and four features and has the characteristics that the output (energy output (ET)) may decrease as ambient temperature (AT) increases. The last one is the Bias correction of numerical prediction model temperature forecast dataset[Cho *et al.*, 2020] which consists of 7750 instances and 23 features and has the characteristics that the output (Next_Tmax) may increase as Present_Tmax increases.

As mentioned in Section 1, the problem that SDL aims to solved is to improve the low generalization performance due to the bias in the data collection. Therefore, in these experiments, separation conditions between training and test datasets are intentionally biased. From Figure 4a and 4b, looking at the entire graph, atemp and cnt are related to strong **positive** correlation, but looking at the only training dataset, atemp and cnt are related to strong **negative** correlation. Similarly, from Figure 4c, the correlation between AT and PE is inverted, and from Figure 4d, the correlation between Next_Tmax and LDAPS_Tmax_lapse is inverted.

Knowledge Table

$Increase$ is set in the sensitivity direction of atemp and temp in the Bike Sharing dataset and of Present_Tmax and LDAPS_Tmax_lapse in the Bias correction dataset, and 15 kinds of $a_{\min}$ candidates (between 0.0 and 1.4 in 0.1 steps) are tested. $Decrease$ is set in the sensitivity direction of AT and RH in CCPP dataset, and 15 kinds of $a_{\max}$ candidates (between 0.0 and -1.4 in -0.1 steps) are tested. The same $a_{\min}$ or $a_{\max}$ values are set for the two features in each dataset.

Model Parameters

Hyperparameters of SDL are the same as experiment 1 except the optimization function is SGD, learning rate is set to 0.1, and mixing ratio set to 1.0. To benchmark SDL performance, as baseline algorithms, three non-shape-constrained algorithms (1) $\sim$ (3) and two shape-constrained algorithms (4), (5) are selected: (1) Linear Regression (LR), (2) Random Forest (RF),(3) Neural Networks (NN), (4) Ensemble Lattice Networks (ELN), and (5) Constrained Linear Regression (CLR). Hyperparameters of RF are used with the default setting of scikit-learn 0.22.1[Pedregosa *et al.*, 2011], and those of NN are the same as SDL. The basic network architecture and learning conditions of ELN are the same as in a lattice tutorial[2]. CLR means linear regression with simple weight capping by using non-negative least squares solver of scipy [Virtanen *et al.*, 2020] so that regression coefficients of the CLR model satisfy coefficient constraints given from knowledge table. For ELN, we tune the 16 kinds of combinations of four kinds of num_lattices G (3, 6, 12, 24) and lattice_rank S (1, int(l/3), int(2l/3), l), and for CLR and SDL, we tune 15 kinds of $a_{\min}$ or $a_{\max}$ in each dataset.

Experimental Results

Table 4 shows root mean squared error (RMSE), coefficient of determination (R2) and training/inference time of each algorithm for each dataset. Each Med., SD, and Avg. were calculated from the results of 5-fold cross validation. The execution environment is Ubuntu 18.04 LTS (40 CPUs (3.00 GHz), 128 GB of Memory). In case of ELN, CLR, and SDL, tuned results, which achieved lowest median RMSE, are displayed, and tuned hyperparameters shown in rows of Param.

From Table 4 and Figure 5, RMSE and R2 of shape-constrained algorithms were significantly better for non-shape-constrained algorithms so that shape constraints are effective when the biased dataset is used. Comparing shape-constrained algorithms, although SDL required the most training time, SDL achieved the best RMSE and R2 in the Bike Sharing and the Bias correction datasets and the second best results in the CCPP dataset, so that it can be said that SDL has the potential to provide the same or better generalization performance than ELN and CLR.

Comparing training times of SDL and NN, since SDL required an additional training time about 12 times on average, this result means that it will take about 6 times more training time to learn one feature that is set a knowledge table than to learn a NN model. Since SDL can be particularly effective in the situation where data collection is biased, the number of the training dataset will be small in practical use. Therefore, we think the training time is unlikely to be a major problem. Nevertheless, if faster learning is needed, it is considered to use neural network techniques that make learning more efficient, or to make SDE calculation once every few iterations.

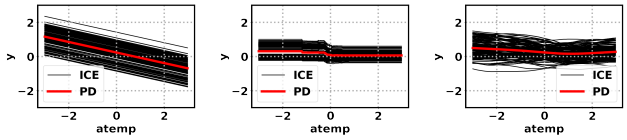
The superiority or inferiority of prediction performances between CLR and SDL tends to be similar to between LR and NN, so that it can be considered that the complexity of the task is related to these performances. In other words, if the task required an expressive power of neural networks, SDL is

[2]https://www.tensorflow.org/lattice/tutorials/shape_constraints/

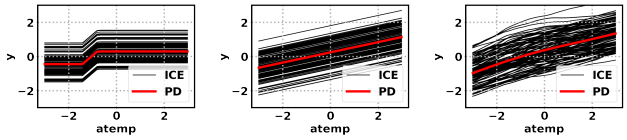
	Bike Sharing (qcut=5)						Bike Sharing (qcut=9)					
	RMSE		R2		Avg. time [sec]		RMSE		R2		Avg. time [sec]	
Algo.	Med.	SD	Med.	SD	Training	Test	Med.	SD	Med.	SD	Training	Test
LR	0.934	0.064	-7.757	1.455	0.001	0.001	1.284	0.037	-5.672	0.776	0.001	0.001
RF	0.969	0.101	-12.64	3.191	0.113	0.006	1.341	0.037	-6.669	1.208	0.101	0.006
NN	0.908	0.088	-5.728	1.951	1.642	0.001	1.282	0.046	-4.996	1.474	1.189	0.001
ELN	0.591	0.056	-0.098	0.124	18.65	1.786	0.621	0.063	-0.176	0.268	23.51	2.458
CLR	0.529	0.062	0.663	0.107	0.004	0.001	0.547	0.047	0.644	0.066	0.004	0.001
SDL	**0.510**	0.054	**0.676**	0.105	22.45	0.001	**0.515**	0.049	**0.708**	0.090	15.72	0.001
Param.	ELN: (G, S)=(6, 7), CLR: $a_{\min}$=0.3, SDL: $a_{\min}$=0.5						ELN: (G, S)=(12, 7), CLR: $a_{\min}$=0.3, SDL: $a_{\min}$=0.5					

	CCPP (qcut=9)						Bias correction (qcut=9)					
	RMSE		R2		Avg. time [sec]		RMSE		R2		Avg. time [sec]	
Algo.	Med.	SD	Med.	SD	Training	Test	Med.	SD	Med.	SD	Training	Test
LR	0.991	0.019	-69.68	8.015	0.001	0.001	1.767	0.032	-3.473	0.087	0.001	0.001
RF	1.149	0.061	-26.00	10.97	0.252	0.019	1.483	0.023	-5.894	0.379	0.780	0.019
NN	1.369	0.063	-9.397	1.653	11.04	0.003	1.481	0.046	-5.589	0.377	11.03	0.003
ELN	0.862	0.034	-19.97	14.21	74.19	1.378	1.019	0.022	-5.417	0.515	56.41	1.776
CLR	**0.328**	0.008	**0.894**	0.003	0.004	0.001	0.794	0.018	-0.058	0.137	0.007	0.001
SDL	0.414	0.031	0.817	0.037	141.59	0.003	**0.759**	0.024	**0.079**	0.087	119.01	0.003
Param.	ELN: (G, S)=(12, 4), CLR: $a_{\max}$=0.5, SDL: $a_{\max}$=0.7						ELN: (G, S)=(3, 15), CLR: $a_{\min}$=0.9, SDL: $a_{\min}$=1.1					

Table 4: Evaluation results of each algorithm for each dataset (5-fold cross validation).

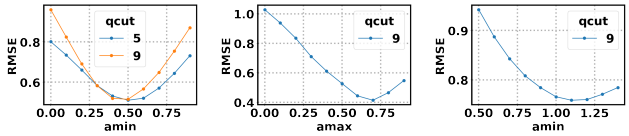


(a) LR (b) RF (c) NN



(d) ELN (e) CLR (f) SDL

Figure 5: PD and ICE plots in the Bike Sharing dataset (qcut=5). Only shape-constrained algorithms (ELN, CLR, and SDL) learned models with correct positive directions.



(a) Bike Sharing (b) CCPP (c) Bias correction

Figure 6: Results of hyperparameter tuning of SDL.

likely to perform better than CLR. When those performances of ELN and SDL are compared, SDL was superior in this experiment. It is considered that this is because ELN does not have hyperparameters equivalent to $a_{\min}$ and $a_{\max}$ (when $a_{\min}$ or $a_{\max}$ were set to 0.3, those performances of SDL tended to equivalent to ELN), so that it can be said that SDL has an advantage if there is domain knowledge of values of coefficient constraints.

Figure 6 shows the change in RMSE for different values of $a_{\min}$ or $a_{\max}$ in each dataset. As can be seen from figures, if there is no knowledge of optimal values of $a_{\min}$ or $a_{\max}$, it is better to set smaller values, such as 0.2, because too large values may adversely affect the performance. Alternatively, it is effective to tune values as in this experiment.

6 Conclusion

This paper proposes Sensitivity Direction Learning (SDL) for learning about a neural network model with user-specified relationships by imposing soft shape constraints which represent domain knowledge. We believe that our research contributes to the field of learning about a neural network model by imposing soft shape constraints. From experimental results, it is considered that SDL can be highly applicable if the following conditions are met: (1) you have domain knowledge about sensitivity directions and especially about values of coefficient constraints, and (2) you expect an expressive power of neural networks. As an example of use cases, we are using SDL to learn about a model for the equipment which handles complex physical phenomena such as combustion, chemical process, etc, and using the trained model for optimization of operating parameters.

Acknowledgements

We would like to thank Satoshi Hara (Osaka University) for his kind guidance on points to keep in mind when writing and submitting papers. And we would like to thank my colleagues, especially Leandro Hideo Iha, for helpful comments.

References

[Amos *et al.*, 2017] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155, 2017.

[Archer and Wang, 1993] Norman P Archer and Shouhong Wang. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 24(1):60–75, 1993.

[Canini *et al.*, 2016] Kevin Canini, Andy Cotter, MR Gupta, M Milani Fard, and Jan Pfeifer. Fast and flexible monotonic functions with ensembles of lattices. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[Cho *et al.*, 2020] Dongjin Cho, Cheolhee Yoo, Jungho Im, and Dong-Hyun Cha. Comparative assessment of various machine learning-based bias correction methods for numerical weather prediction model forecasts of extreme air temperatures in urban areas. *Earth and Space Science*, 7(4):e2019EA000740, 2020.

[Daniels and Velikova, 2010] Hennie Daniels and Marina Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

[Dua and Graff, 2017] Dheeru Dua and Casey Graff. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2017.

[Fanaee-T and Gama, 2014] Hadi Fanaee-T and Joao Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2(2-3):113–127, 2014.

[Friedman, 2001] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[Gill and Hall, 2019] Navdeep Gill and Patrick Hall. *An Introduction to Machine Learning Interpretability, 2nd Edition*. O'Reilly Media, Inc., October 2019.

[Goldstein *et al.*, 2015] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

[Gupta *et al.*, 2018] Maya Gupta, Dara Bahri, Andrew Cotter, and Kevin Canini. Diminishing returns shape constraints for interpretability and regularization. In *Advances in neural information processing systems*, pages 6834–6844, 2018.

[Gupta *et al.*, 2019] Akhil Gupta, Naman Shukla, Lavanya Marla, and Arinbjörn Kolbeinsson. Monotonic trends in deep neural networks. *arXiv preprint arXiv:1909.10662*, 2019.

[Horváth *et al.*, 2019] K Horváth, B van Esch, D Vreeken, I Pothof, and J Baayen. Convex modeling of pumps in order to optimize their energy use. *Water Resources Research*, 55(3):2432–2445, 2019.

[McMenamin, 2008] J Stuart McMenamin. Defining normal weather for energy and peak normalization. *Itron Forecasting, available at: www. itron. com/PublishedContent/DefiningNormalWeatherfor EnergyandPeakNormalization. pdf*, 2008.

[Molnar, 2019] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2019.

[Mulders *et al.*, 2020] Sebastiaan Paul Mulders, Tobias Gybel Hovgaard, Jacob Deleuran Grunnet, and Jan-Willem van Wingerden. Preventing wind turbine tower natural frequency excitation with a quasi-lpv model predictive control scheme. *Wind Energy*, 23(3):627–644, 2020.

[Nguyen and Howe, 1992] Clark TC Nguyen and Roger T Howe. Quality factor control for micromechanical resonators. In *Tech. Dig. IEEE Int. Electron Devices Meeting*, pages 14–16, 1992.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Sill, 1998] Joseph Sill. Monotonic networks. In *Advances in neural information processing systems*, pages 661–667, 1998.

[Tüfekci, 2014] Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

[Virtanen *et al.*, 2020] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[Wehenkel and Louppe, 2019] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. In *Advances in Neural Information Processing Systems*, pages 1545–1555, 2019.

[You *et al.*, 2017] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. Deep lattice networks and partial monotonic functions. In *Advances in neural information processing systems*, pages 2981–2989, 2017.